Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2011

M. Petit-Huguenin
Stonyfish, Inc.
March 13, 2011

Configuration of Access Control Policy in REsource LOcation And Discovery (RELOAD) Base Protocol
draft-petithuguenin-p2psip-access-control-01

Abstract

This document describes an extension to the REsource LOcation And Discovery (RELOAD) base protocol to distribute the code of new Access Control Policies without having to upgrade the RELOAD implementations in an overlay.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The RELOAD base protocol specifies an Access Control Policy as "defin[ing] whether a request from a given node to operate on a given value should succeed or fail." The paragraph continues saying that "[i]t is anticipated that only a small number of generic access control policies are required", but there is indications that this assumption will not hold. On all the RELOAD Usages defined in other documents than the RELOAD base protocol, roughly 50% defines a new Access Control Policy.

The problem with a new Access Control Policy is that, because they are executed when a Store request is processed, they need to be implemented by all the peers and so require an upgrade of the software. This is something that is probably not possible in large overlays or on overlays using different implementations. For this reason, this document proposes an extension to the RELOAD configuration document that permits to transport the code of a new Access Control Policy to each peer.

This extension defines a new element <access-control-code> that can be optionally added to a <kind> element in the configuration document. The <access-control-code> element contains ECMAScript [ECMA-262] code that will be called for each StoredData object in a StoreReq processed by a peer. The code receives four parameters, corresponding to the Resource-ID, Signature, Kind and StoredDataValue of the value to store. The code returns true or false to signal to the implementation if the request should succeed or fail.

For example the USER-MATCH Access Control Policy defined in the base protocol could be redefined by inserting the following code in an <access-control-code> element:

```
return resource.equalsHash(signature.user_name.bytes());
```

The <kind> parameters are also passed to the code, so the NODE-MULTIPLE Access Control Policy could be implemented like this:

```
for (var i = 0; i < kind.params['max-node-multiple']; i++) {
    if (resource.equalsHash(signature.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Processing an extended Kind

A peer receiving a <kind> definition, either by retrieving it from the configuration server or in a ConfigUpdateReq message, MUST verify the signature in the kind-signature element before executing the code.

If the <access-control-code> element is present in the namespace allocated to this specification, and the Access Control Policy is not natively implemented, then the code inside the element MUST be called for each DataValue found in a received StoreReq for this Kind. For each call to the code, the following ECMAScript objects, properties and functions MUST be available:

resource: An opaque object representing the Resource-ID, as an array of bytes.

resource.equalsHash(Object...): Returns true if hashing the concatenation of the arguments according to the mapping function of the overlay algorithm is equal to the Resource-ID. Each argument is an array of bytes.

signature.user_name: The rfc822Name stored in the certificate that was used to sign the request, as a String object.

signature.node_id: The Node-ID stored in the certificate that was used to sign the request, as an array of bytes.

kind.id: The id of the Kind associated with the entry, as a Number object.

kind.name: The name of the Kind associated with the entry, as a String object.

kind.data_model: The name of the Data Model associated with the entry, as a String object.

kind.access_control: The name of the Access Control Policy associated with the entry, as a String object.

kind.params: An associative array containing the parameters of the Access Control Policy as specified in the configuration file.

- max-count: The value of the max-count element in the configuration file, as a String object.
- max-size: The value of the max-size element in the configuration file as a String object.

max-node-multiple: If the Access Control is MULTIPLE-NODE, contains the value of the max-node-multiple element in the configuration file, as a String object. If not, this property is undefined.

entry.index: If the Data Model is ARRAY, contains the index of the entry, as a Number object. If not, this property is undefined.

entry.key: If the Data Model is DICTIONARY, contains the key of the entry, as an array of bytes. If not, this property is undefined.

entry.storage_time: The date and time used to store the entry, as a Date object.

entry.lifetime: The validity for the entry in seconds, as a Number object.

entry.exist: Indicates if the entry value exists, as Boolean object.

entry.value: This property contains an opaque object that represents the whole data, as an array of bytes.

The properties SHOULD NOT be modifiable or deletable and if they are, modifying or deleting them MUST NOT modify or delete the equivalent internal values (in other words, the code cannot be used to modify the elements that will be stored).

If addition to the "max-count", "max-size" and eventual "max-node-multiple" properties in the kind.params associative array, any extension element in any namespace found in the <kind> element MUST be added to this array, using the element name as key and the content as value.

The value returned by the code is evaluated to true or false, according to the ECMAScript rules. If the return value of one of the call to the code is evaluated to false, then the StoreReq fails, the state MUST be rolled back and an Error_Forbidden MUST be returned.

4. Security Considerations

TBD

5. IANA Considerations

No IANA considerations.

6. Acknowledgements

This document was written with the xml2rfc tool described in [RFC2629].

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[I-D.ietf-p2psip-base] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-12 (work in progress), November 2010.

[ECMA-262] Ecma, "ECMAScript Language Specification 3rd Edition", December 2009.

7.2. Informative References

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[I-D.ietf-p2psip-service-discovery] Maenpaa, J. and G. Camarillo, "Service Discovery Usage for REsource LOcation And Discovery (RELOAD)", draft-ietf-p2psip-service-discovery-02 (work in progress), January 2011.

[I-D.knauf-p2psip-disco] Knauf, A., Hege, G., Schmidt, T., and M. Waehlisch, "A RELOAD Usage for Distributed Conference Control (DisCo)", draft-knauf-p2psip-disco-01 (work in progress), December 2010.

Appendix A. Examples

A.1. Standard Access Control Policies

This section shows the ECMAScript code that could be used to implement the standard Access Control Policies defined in [I-D.ietf-p2psip-base].

A.1.1. USER-MATCH

```
   String.prototype['bytes'] = function() {
       var bytes = [];
       for (var i = 0; i < this.length; i++) {
           bytes[i] = this.charCodeAt(i);
       }
       return bytes;
   };

   return resource.equalsHash(signature.user_name.bytes());
```

A.1.2. NODE-MATCH

```
   return resource.equalsHash(signature.node_id);
```

A.1.3. USER-NODE-MATCH

```
   String.prototype['bytes'] = function() {
       var bytes = [];
       for (var i = 0; i < this.length; i++) {
           bytes[i] = this.charCodeAt(i);
       }
       return bytes;
   };

   var equals = function(a, b) {
       if (a.length !== b.length) return false;
       for (var i = 0; i < a.length; i++) {
           if (a[i] !== b[i]) return false;
       }
       return true;
   };

   return resource.equalsHash(signature.user_name.bytes())
     && equals(entry.key, signature.node_id);
```

A.1.4. NODE-MULTIPLE

```
Number.prototype['width'] = function(w) {
    var bytes = [];
    for (var i = 0; i < w; i++) {
        bytes[i] = (this >>> ((w - i - 1) * 8)) & 255;
    }
    return bytes;
};

for (var i = 0; i < kind.params['max-node-multiple']; i++) {
    if (resource.equalsHash(signature.node_id, i.width(4))) {
        return true;
    }
}
return false;
```

A.2. Service Discovery Usage

[I-D.ietf-p2psip-service-discovery] defines a specific Access Control Policy (NODE-ID-MATCH) that need to access the content of the entry to be written. If implemented as specified by this document, the <kind> element would look something like this:

```
<kind name='REDIR'
  xmlns:acp='http://implementers.org/access-control-policy'
  xmlns:ext='http://implementers.org/my-ext'>
    <data-model>DICTIONARY</data-model>
    <access-control>NODE-ID-MATCH</access-control>
    <max-count>100</max-count>
    <max-size>60</max-size>
    <ext:branching-factor>2</ext:branching-factor>

    <acp:access-control-code>
        /* Insert here the code from
           http://jsfromhell.com/classes/bignumber
         */

       var toBigNumber = function(node_id) {
           var bignum = new BigNumber(0);
           for (var i = 0; i < node_id.length; i++) {
               bignum = bignum.multiply(256).add(node_id[i]);
           }
           return bignum;
       };

       var checkIntervals = function(node_id, level, node, factor) {
           var size = new BigNumber(2).pow(128);
           var node = toBigNumber(node_id);
           for (var f = 0; f < factor; f++) {
               var temp = size.multiply(new BigNumber(f)
                 .pow(new BigNumber(level).negate()));
               var min = temp.multiply(node.add(new BigNumber(f)
                 .divide(factor)));
               var max = temp.multiply(node.add(new BigNumber(f + 1)
                 .divide(factor)));
               if (node.compare(min) === -1 || node.compare(max) == 1
                 || node.compare(max) == 0) return false;
           }
           return true;
       };

       var equals = function(a, b) {
           if (a.length !== b.length) return false;
           for (var i = 0; i < a.length; i++) {
               if (a[i] !== b[i]) return false;
           }
           return true;
       };

       var level = function(value) {
           var length = value[16] * 256 + value[17];
           return value[18 + length] * 256 + value[18 + length + 1];
       };

       var node = function(value) {
           var length = value[16] * 256 + value[17];
           return value[18 + length + 2] * 256
             + value[18 + length + 3];
       };

       var namespace = function(value) {
           var length = value[16] * 256 + value[17];
           return String.fromCharCode(value.slice(18, length));
       };

       return equals(entry.key, signature.node_id)
         && (!entry.exists || checkIntervals(entry.key,
           level(entry.value), node(entry.value),
           kind.params['branching-factor']))
         && (!entry.exists
           || resource.equalsHash(namespace(entry.value),
             level(entry.value), node(entry.value)));
    </acp:access-control-code>
</kind>
```

Note that the code for the BigNumber object was removed from this example, as the licensing terms are unclear. The code is available at <http://jsfromhell.com/classes/bignumber>.

The <branching-factor> parameter is used to match the <redirBranchingFactor> parameter that is not accessible to the code. The signer of the kind must be sure that the two match. In fact the branching factor could have been set directly in the code, but that would make it more difficult to change.

Appendix B. Release notes

This section must be removed before publication as an RFC.

B.1. Modifications between -01 and -00

- Changed reference from JavaScript to ECMAScript.
- Changed signature from equals() to equalsHash().
- Fixed the examples following implementation.
- Replaced automatic decoding of value by ECMAScript code.
- Added the type of each property.
- Specified that the code cannot be used to modify the value stored.

B.2. TODO List

- Need to present the complete list of certificates for the DisCo [I-D.knauf-p2psip-disco] Usage USER-CHAIN-MATCH.

Author’s Address

Marc Petit-Huguenin
Stonyfish, Inc.

Email: petithug@acm.org